ITEN ENGINEERING
Software Development

Introduction to python™

Ausgabe 10.2022

1

Content

2

Chapter 1

Kick Start

What is Python?

- Python is an interpreted, interactive, object-oriented programming language.
- It incorporates modules, exceptions, dynamic typing, very high level dynamic data types, and classes.
- It supports multiple programming paradigms beyond object-oriented programming, such as procedural and functional programming.
- Python combines remarkable power with very clear syntax.

Quelle: https://docs.python.org/3/faq/general.html

What is Python?

- It has interfaces to many system calls and libraries, as well as to various window systems, and is extensible in C or C++.
- It is also usable as an extension language for applications that need a programmable interface.
- Finally, Python is portable: it runs on many Unix variants including Linux and macOS, and on Windows.

Quelle: https://docs.python.org/3/faq/general.html

5

Compiled vs. interpreted

- Source code is compiled before execution
- Generates executable file for specific architecture and operating system
- Execution of the program is fast

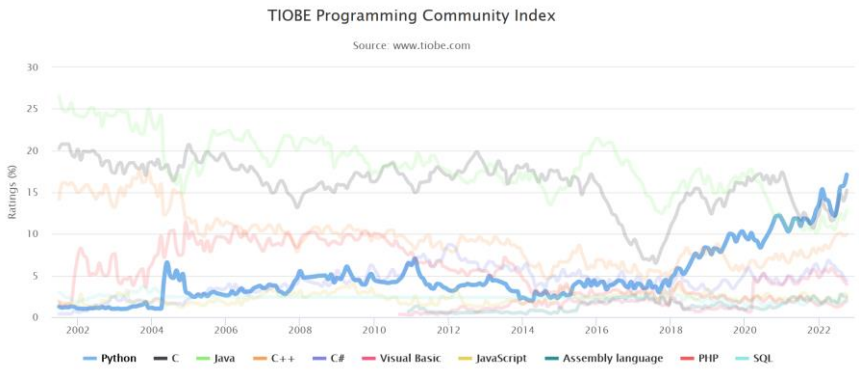
- Source code is interpreted and executed during runtime
- Execution is slower
- Interactive programming is possible

6

Why Python?

- Interpreted languages are ideal for interactive works, research, etc.
- Simple syntax
- Powerful libraries
- Very popular in the field of Data Science (Artificial Intelligence, Machine Learning)
- Interactive work with Jupiter Notebooks
- Powerful graphics libraries
- Freely available (Open Source)
- Extensible and embeddable (C, C++)

Why Python?



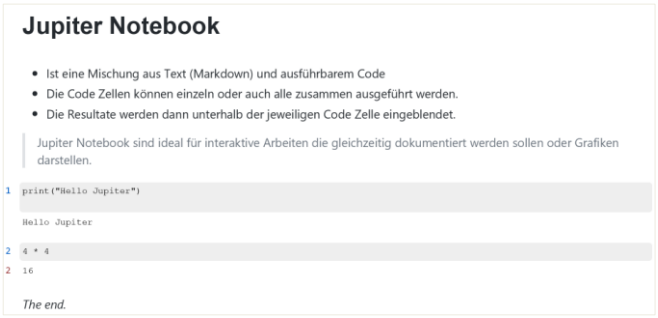
Quelle: https://www.tiobe.com/tiobe-index

Demo

- hello.py

```
print("Hello World")
```

- hello.ipynb



Exercises

Chapter 1. Kickstart

- ❑ HelloWorld
- ❑ HelloJupiter

Chapter 2

Basics

Section I

Operators, Variables, Comments, Help & Type Information

Operators

Python supports the following types of operators:

- Arithmetic Operators
- Comparison (Relational) Operators
- Assignment Operators
- Logical Operators
- Bitwise Operators
- Membership Operators
- Identity Operators

Operators II

Operator	Description	Sample	
+x, -x	Sign	-3	
+, -	Addition, subtraction	10 – 3	Ergebnis: 7
*, /, %	Multiplication, division, rest	27 % 7	Ergebnis: 6
//	Integer division	7//2	Ergebnis: 3
**	Potency	10 ** 3	Ergebnis: 1000
+=, -=, *=, /=, **=	Compound Operators	x += 3	Ergebnis: x = x + 3
or, and, not	Boolean OR, AND, NOT	(a or b) and c	
in	Element in set	1 in [0, 1]	Ergebnis: True
<, <=, >, >=, !=, ==	Comparison operators	2 <= 3	Ergebnis: False
is, is not	Comparison objects	x=5; y=x; x is y	Ergebnis: True
|, &, ^, ~x	Bitwise OR, AND, XOR, NOT	6 ^ 3	Ergebnis: 4
<<, >>	Shift operators	6 << 2	Ergebnis: 24

Details: https://docs.python.org/3/reference/lexical_analysis.html#operators

Operator Precedence

Operator	Description
`:=`	Assignment expression
`lambda`	Lambda expression
`if – else`	Conditional expression
`or`	Boolean OR
`and`	Boolean AND
`not x`	Boolean NOT
`in, not in, is, is not, <, <=, >, >=, !=, ==`	Comparisons, including membership tests and identity tests
`|`	Bitwise OR
`^`	Bitwise XOR
`&`	Bitwise AND
`<<, >>`	Shifts
`+, -`	Addition and subtraction
`*, @, /, //, %`	Multiplication, matrix multiplication, division, floor division, remainder [5]
`+x, -x, ~x`	Positive, negative, bitwise NOT
`**`	Exponentiation [6]
`await x`	Await expression
`x[index], x[index:index], x(arguments...), x.attribute`	Subscription, slicing, call, attribute reference
`(expressions...), [expressions...], {key: value...}, {expressions...}`	Binding or parenthesized expression, list display, dictionary display, set display

Quelle: https://docs.python.org/3/reference/expressions.html?highlight=precedence

15

Variables

```
counter = 100           # Integer
miles   = 1000.0        # Floating point
name    = "John"        # String (instanziert vom Literal "John")
age     = str(54)       # String (instanziert via Konstruktor)

print("counter = ", counter)
print("miles   = ", miles)
print("name    = ", name)
print("age     = ", age)
```

Output

```
counter = 100
miles   = 1000.0
name    = John
Age     = 54
```


16

Static vs. dynamic type declaration

- Languages like C or Java have a static type declaration.
  - That means the type of a variable can not change during runtime
  - Only the value of the variable can change
- Python assigns the type of a variable dynamically
  - There is no type specification during declaration
  - Both the value and the type of a variable can change at runtime

Dynamic type assignment

```
i = 42
print(type(i))

i = "Hallo"
print(type(i))

i = [3,9,17]
print(type(i))
```

Output

```
<class 'int'>
<class 'str'>
<class 'list'>
```

Data types

Type	Class
Text	str
Numeric	int, float, complex
Sequence	list, tuple, range
Map	dict
Set	set, frozenset
Boolean	bool
Binary	bytes, bytearray, memoryview

Display & check data type

```
solution = 42
print(type(solution))
print(isinstance(solution, int))
```

Output

```
<class 'int'>
True
```


Casting

```
s = "12.5"
print(type(s))
print(s)

f = float(s)
print(type(f))
print(f)

i = int(s)
```

Output

```
<class 'str'>
12.5
<class 'float'>
12.5
Traceback (most recent call last):
  File ".../02-basics/casting.py", line 9, in <module> i = int(s)
ValueError: invalid literal for int() with base 10: '12.5'
```

Variables names

- Must start with letter or underscore "_".
- The other characters may consist of any sequence of letters, digits and the underscore.
- Variable names are case-sensitive:
  This means that Python is case sensitive.
- No reserved words

Reserved words

- if, elif, else
- True, False, None
- and, or, not
- for, while, continue, break
- try, except, finally, raise
- def, class, lambda
- return, yield, pass
- from, import, as, with
- is, in, assert
- del, global, nonlocal

Comments

```python
def test_elapsed(self):
    # init
    sw = StopWatch()
    # test
    sw.start()
    time.sleep(3)
    sw.stop()
    # debug
    if (self.debug):
        print("Test StopWatch:")
        print("Start  ", sw.start_time)
        print("Stop   ", sw.stop_time)
        print("Elapsed", sw.elapsed())
    # check
    self.assertTrue(sw.stop_counter - sw.start_counter >= 2.9) # give some tolerance
```

Bis zum Ende der Zeile

Comments II

```python
class ReportSystemPerformance(AbstractReport):
    """Report the system performance of the models.

    Sample Output
    -------------
    ReportSystemPerformance
                          Time    CPU  Memory Peak
    LangDetect            4.31   8.33       499253
    LangDetectSpacy      12.83   8.33      1336925
    LangFromStopwords     0.10   8.31       495957
    LangFromChars        65.28  17.18    282294433
    AzureTextAnalytics  146.70   1.18       594964
    Time elapsed: 700.58 sec

    Report saved to: outcome/ReportSystemPerformance.csv
    """

    def __init__(self):
        AbstractReport.__init__(self, "ReportSystemPerformance")
        self.model = None
        self.csv_file = "articles_all_1k.csv"
```

Mehrere Zeilen

Comments III

```
def predict(self, text):
    """Predict the language code for the given text.
    Args:
        text (str): The text to predict the language of.
    Returns:
        str: The language code (ISO-639-1)
    """
    raise NotImplementedError("The method is not implemented yet.")
```

Beispiel Methode

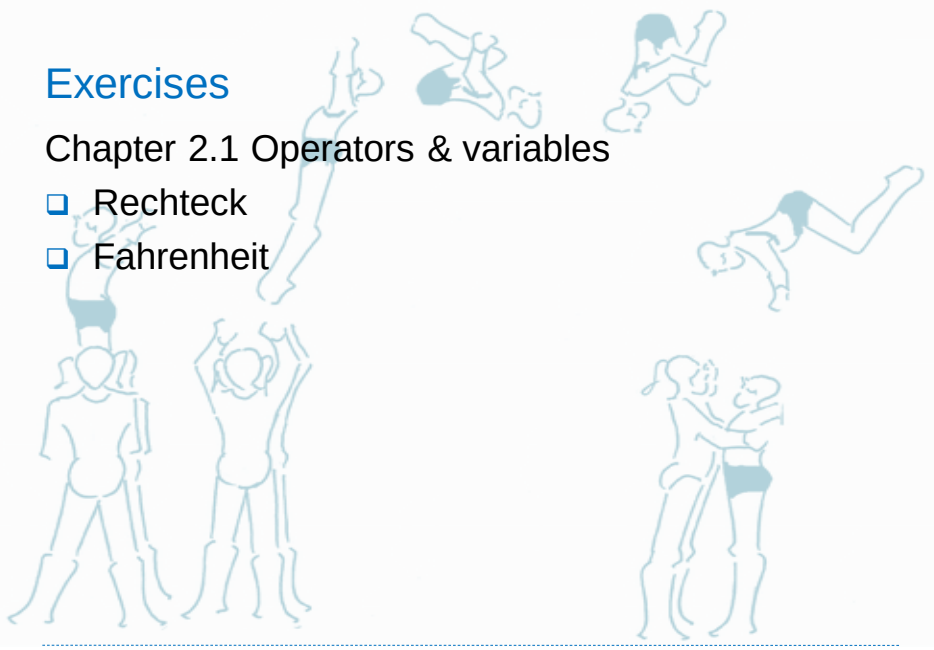
Help and Type Information

```
obj = str
help(obj)           # show help of this object / function
print(obj)          # print object to default output
type(obj)           # show type of object

dir()               # List all variables of the namespace
dir(obj)            # List all attributes of object instance

c = complex         # class
c.__dict__          # show attributes of class
vars(c)             # show attributes of class

class Foo(object):  # show instance attributes
  def __init__(self):
    self.a = 1
    self.b = 2

vars(Foo())         #==> {'a': 1, 'b': 2}
vars(Foo()).keys()  #==> ['a', 'b']
```

Exercises

Chapter 2.1 Operators & variables

- Rechteck
- Fahrenheit



Section II

Control structures

if – elif – else

```
seq = [1,2,3]

if len(seq) == 0:
   print("sequence is empty")
elif len(seq) == 1:
   print("sequence contains one element")
else:
   print("sequence contains several elements“)
```

Output

```
sequence contains several elements
```

Note:

- Code blocks are defined by indentations
- No brackets are used!

if shorthand (Ternary Operator)

```
a = 5
b = 3

x = 10 if a > b else 1    # better readable

y = a > b and 10 or 1    # style more like java or other languages

print(x)
print(y)
```

Output

```
10
10
```

for loop

```
# For loop over list of words
words = ["winter“, "is“, "coming"]
for word in words:
   print(word)
```

Output

```
winter
is
coming
```

For Loop Control:

- With break the for loop is left
- With continue the next iteration of the for loop is executed

for loop in reverse order

```
# For loop in reverse order
words = ["coming","is","winter"]
for word in reversed(words):
   print(word)
```

Output

```
winter
is
coming
```

for loop over multiple arrays with zip

```
# For loop over mutliple arrays (zip)
names = ["Peter", "Jane", "Fred"]
ages = [31, 35, 4]

for t in zip(names, ages):                  # use a tuple
   print(t)

for name, age in zip(names, ages):          # use multiple loop-variables
   print(name, "is", age)

for i, name in enumerate(names):            # use enumerate and index
   print(name, "is", ages[i])

for i in range(len(names)):                 # use range and index
   print(names[i], "is", ages[i])
```


for loop with range and enumerate

```
# For loop with range
for i in range(10):                 # i = 0..9
   print(i)

for i in range(5,10):               # i = 5..9
   print(i)

for i in range(0,100,10):           # i = 0, 10, 20, ... 80, 90
   print(i)

# For loop with index and element (enumerate)
list = ["first", "second"]
for pair in enumerate(list):
   print(type(pair), pair)          # <class 'tuble'> (0, 'first')
                                    # <class 'tuble'> (1, 'second')
```

Range (start, end, step)
- Startwert (inklusiv),
- Endwert (exklusiv),

List Comprehension

```
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]
newlist = []

for x in fruits:
  if "a" in x:
    newlist.append(x)

# List Comprehension
fruits = ["apple", "banana", "cherry", "kiwi", "mango"]

newlist = [x for x in fruits if "a" in x]
```

Conventional for Loop for creating a new list, containing all elements with an “a”

Same Result with List Comprehension

- List comprehension offers a shorter syntax (than the for loop) when you want to create a new list based on the values of an existing list.

while

```
data = [1, 5, 4, 3, 4, 1, 8]
i = 0

sum = 0
count = 0

while (i < len(data)):
   value = data[i]
   sum = sum + value
   count = count + 1
   i = i+1;

print(data)
print(sum)
```

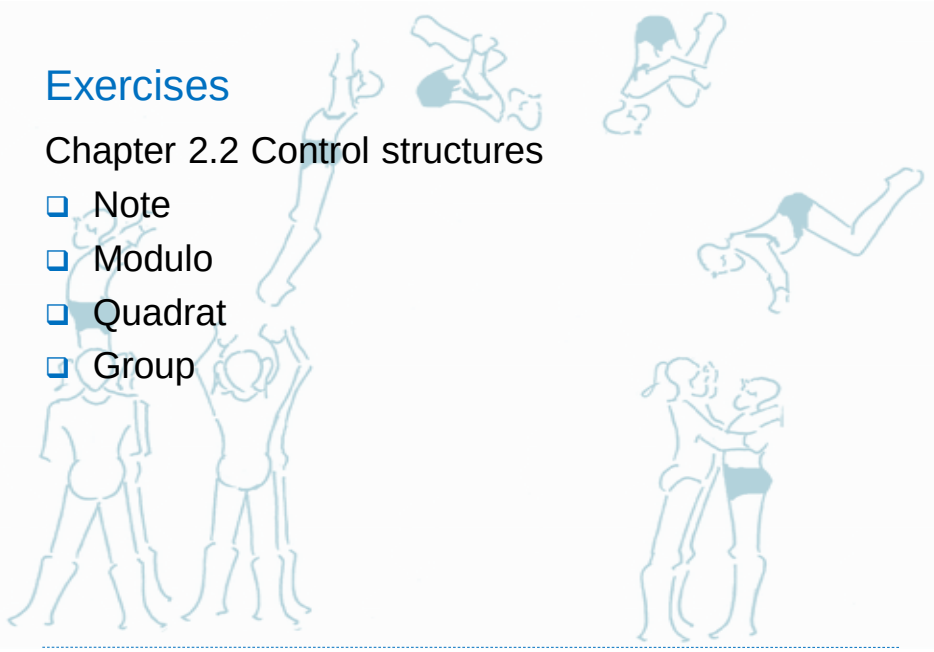
Output

```
[1, 5, 4, 3, 4, 1, 8]
26
```

Exercises

Chapter 2.2 Control structures

- Note
- Modulo
- Quadrat
- Group



Section III

Error handling with exceptions

Error handling with exception

- In case of an error (exception) Python normally stops the execution and prints an error message
- If you want to handle the errors in the code, you can do this with:
  - try Start of statements that can throw an exception
  - except "Catching" the exception and instructions for error handling
  - finally Completion Block with instructions that are executed in both good and error cases

try – except

```
try:
   input = "12x"
   nr = int(input)
except:
   print("Invalid input: ", input)

try:
   input = "12x"
   nr = int(input)
except Exception as e:
   print("Invalid input: ", str(e) )
```

Output

```
Invalid input:  12x
Invalid input:  invalid literal for int() with base 10: '12x'
```

try – except II

```
import sys

try:
   f = open('myfile.txt')
   s = f.readline()
   i = int(s.strip())
except OSError as err:
   print("OS error: {0}".format(err))
except ValueError:
   print("Could not convert data to an integer.")
except (RuntimeError, TypeError, NameError):
   print("Catch all other expected errors.")
except:
   print("Unexpected error:", sys.exc_info()[0])
   raise
```

Specific error handling

Catch multiple errors within the same error handling

Handle all remaining errors and continue to propagate with raise!

Output

```
OS error: [Errno 2] No such file or directory: 'myfile.txt'
```

Details: https://docs.python.org/3/tutorial/errors.html

try – except – else – finally

```
def divide(x, y):
   success = True
   try:
      result = x / y
   except ZeroDivisionError:
      success = False
      print("division by zero!")
   else:
      print("result is", result)
   finally:
      if (success):
         print("> divison successfully done")
      else:
         print("> division failed")

divide(10,2)
divide(10,0)
divide(10,5)
```

Will be executed only if try block ends without error

Always executed

Output

```
result is 5.0
> divison successfully done
division by zero!
> division failed
result is 2.0
> divison successfully done
```

raise

```
try:
   input = -1
   if input < 0:
      raise Exception("Number must by positive")
except Exception as e:
   print("Invalid input:", str(e))
```

Output

```
Invalid input: Number must by positive
```

- The following **Built-in Exceptions** are available
  - https://docs.python.org/3/library/exceptions.html
- You can also create your own exceptions
  - Details see **User-defined Exceptions** https://docs.python.org/3/tutorial/errors.html

Exercises

Chapter 2.3 Exceptions

- ❑ Validation

Section IV

Functions & lambda

Functions

```
def hello():
   print("Hello")

def greeting(name):
   print("Hello", name)

def weekend_greeting(name, greeting):
   print("Hello %s, i wish you %s"%(name, greeting))

hello()
greeting("Tom")
weekend_greeting("Zoé", "a nice weekend")
```

Output

```
Hello
Hello Tom
Hello Zoé, i wish you a nice weekend
```

Functions with return value

```
def add(a, b):
   return a + b

def mean(values):
   return sum(values) / len(values)

res = add(3,9)
print ("add(3,9) =", res)

res = mean([4,8,12])
print ("mean([4,8,12]) =", res)
```

Output

```
add(3,9) = 12
mean([4,8,12]) = 8.0
```


Positional and Keyword Arguments

```
def f1(arg1, arg2, arg3):
   print("f1: arg1={}, arg2={}, arg3={}".format(arg1,arg2,arg3))

def f2(arg1=None, arg2=10, arg3="Default"):
   print("f2: arg1={}, arg2={}, arg3={}".format(arg1,arg2,arg3))

def f3(arg1, arg2, arg3="Default", arg4=99):
   print("f3: arg1={}, arg2={}, arg3={}, arg4={}".format(arg1,arg2,arg3,arg4))

f1(1,2,3)
f2()
f2(arg2=22)
f3(1,2)
f3(1,2, arg4=88)
```

«Positional» Argumente sind obligatorisch und werden durch ihre Position zugeordnet

«Keyword» Argumente sind fakultativ und werden durch Position oder Name zugeordnet

«Mixed» Argumente werden durch die Position und Namen zugeordnet

Output

```
f1: arg1=1, arg2=2, arg3=3
f2: arg1=None, arg2=10, arg3=Default
f2: arg1=None, arg2=22, arg3=Default
f3: arg1=1, arg2=2, arg3=Default, arg4=99
f3: arg1=1, arg2=2, arg3=Default, arg4=88
```

- Further possibilities see appendix
  - Arguments "unpack" with *args and **kwargs
  - variable number of arguments with *args und **kwargs

Lambda

- A lambda function is a small anonymous function
  - with any number of arguments
  - and an expression
- Syntax:
  lambda arguments **:** expression

```
add = lambda a, b : a + b

print(add(1,2))
```

```
3
```
Output

Lambda II

- Lambda are often used in connection with the processing of list elements (e.g. with map or filter).
- A lambda function can also be returned as the result of another function
- This allows elegant and powerful functions as the following example shows:

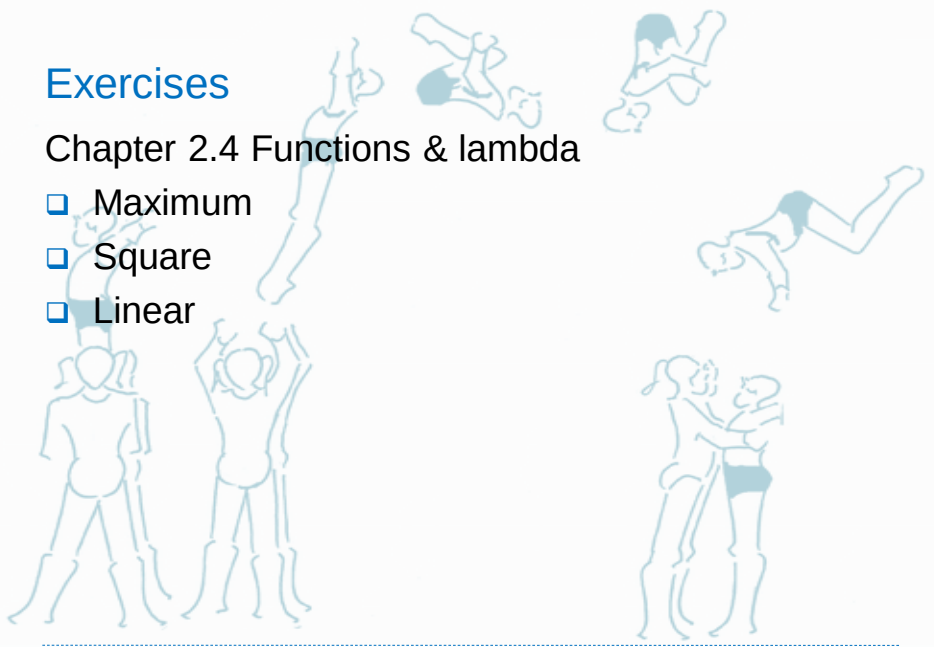
```
def multiplier(n):
  return lambda a : a * n

double = multiplier(2)
triple = multiplier(3)

print(double(10))
print(triple(10))
```

```
20
30
```
Output

Exercises

Chapter 2.4 Functions & lambda

- Maximum
- Square
- Linear



Chapter 3

Data types

Section I

Strings

Declaration

- Python 3 strings are coded in Unicode
- There are various possibilities for the declaration of Strings:

```
quote          = '"Titanic" is a cool movie.'
doublequote    = "'Titanic' is a cool movie"
escape = 'I\'m hungry'
multilines = """This is going
over multiples lines"""

path = "c:\\tmp"
rawpath = r"c:\tmp"                # Raw String - Escape Sequenzen werden ignoriert
unicode = '\U000000e4'             # ä (Python 3, wird direkt als Unicode interpretiert)
unicodePython2 = u'\U000000e4'     # ä (Python 2)
```

capitalize, format String

```
# capitalize
s = "this is a test"
print(s.capitalize())                                   # This is a test
# format
name = "Tom"
s = "Hello {}".format(name)                             # Hello Tom
print(s)
s = "i have {} {}".format(1,"cat")                      # i have 1 cat
print(s)
s = "i have {1} {0}".format("cat",1)                    # i have 1 cat
print(s)
s = "i have {count} {animal}".format(count=1, animal="cat")   # i have 1 cat
print(s)
```

Output

```
This is a test
Hello Tom
i have 1 cat
i have 1 cat
i have 1 cat
```

Weitere Beispiele: https://pyformat.info

format Number

```
s = "pi={:.2}".format(pi)              # {:breite.genauigkeit}
print("[", s, "]", sep="")
s = "pi={:10.2}".format(pi)
print("[", s, "]", sep="")

                                       # {ausrichtung:breite.genauigkeit}
s = "pi={:>10.2}".format(pi)           # rechts
print("[", s, "]", sep="")
s = "pi={:<10.2}".format(pi)           # links
print("[", s, "]", sep="")
s = "pi={:^10.2}".format(pi)           # centriert
print("[", s, "]", sep="")

s = "pi={:0=10.2}".format(pi)          # {füllung=:breite.genauigkeit}
print("[", s, "]", sep="")
```

Output

```
[pi=3.1]
[pi=       3.1]
[pi=       3.1]
[pi=3.1       ]
[pi=   3.1    ]
[pi=00000003.1]
```

Weitere Beispiele: https://pyformat.info

find, rfind, replace, starts/endswith, in

- s.find(…) return first position (index) of substring
- s.rfind(…) return last position (index) of substring

```
text = "this and that"
text.find("th")                 # 0
text.rfind("th")                # 9

s = "Fall is coming".replace("Fall", "Winter")  # Winter is coming

s.startswith("Winter")          # True
s.startswith("Fall")            # False

s.endswith("coming")            # True
s.endswith("leaving")           # False

"test" in "this is a test"      # True
"x" in "abc"                    # False
```


split, join

```
text = "Hello Tom, how are you"

tokens = text.split()           # ["Hello", "Tom,", "how", "are", "you"]
print(tokens)

tokens = text.split(",")        # ["Hello Tom", "how are you"]
print(tokens)

num = 15.75
before, after = str(num).split(".")
print(before, ":", after)

tokens = ["Today", "we", "have", "a", "Python", "course."]
text = " ".join(tokens)
print(text)
```

Output

```
['Hello', 'Tom,', 'how', 'are', 'you']
['Hello Tom', ' how are you']
15 : 75
Today we have a Python course.
```

strip, lstrip, rstrip, ljust, center, rjust

```
# strip, lstrip, rstrip
"  some text  ".strip()          # "some text"
"_some text_".strip("_")         # "some text"
"  some text  ".lstrip()         # "some text  "
"  some text  ".rstrip()         # "  some text"

# ljust, center, rjust
s = "expand text".ljust(20)
print("[", s, "]", sep="")       # [expand text       ]

s = "expand text".rjust(20)
print("[", s, "]", sep="")       # [       expand text]

s = "expand text".center(20)
print("[", s, "]", sep="")       # [   expand text    ]
```

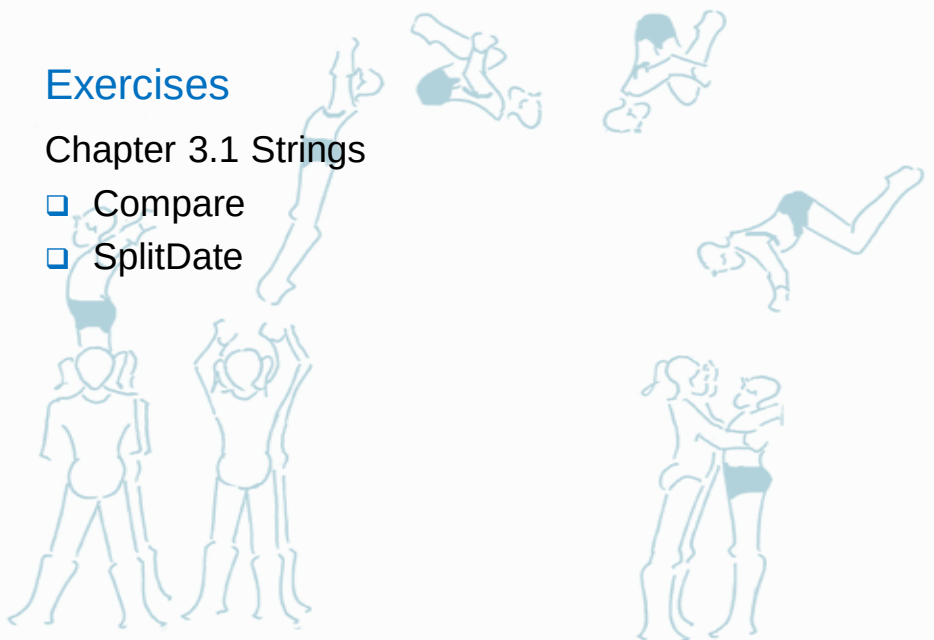
Output

```
[expand text          ]
[          expand text]
[    expand text      ]
```


Exercises

Chapter 3.1 Strings

- Compare
- SplitDate



Section II

List & Tuples

List & Tuples

- Tuples are immutable, the values cannot be changed, the list size is fixed
- Lists are mutable, values and the number of entries of a list can be changed dynamically
- Set and dictionaries are also mutable

```
# Mutable List (Set, Dict):
l = [1,2,3]          # List class list() is mutable
l[1] = 22            # works fine

# Immutable Tuple:
t = (1,2,3)          # Tuple class tuple() is immutable
t[1] = 22            # raises TypeError
```

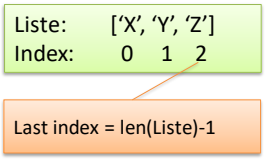
Mnemonic for mutable: LSD (List, Set, Dictionaries)

Mutable vs. Immutable

- Modifiable objects are modified "in place", which affects all variables pointing to the object.
  - To modify the elements of a list during a loop, you should access the element via index
  - A list that is iterated over should not be modified at the same time.
- For immutable objects a new object must be created (each time a new value is assigned to a variable)
  - Immutable objects are not modified during iteration
  - If you modify an immutable object, you iterate over, it creates a new object and has no influence on the loop

Initialization, len, min, max, count, index

```
# Initialization
l1 = []                 # []
l2 = [1,2,3]            # [1, 2, 3]
l3 = list("abc")        # ['a', 'b', 'c']
l4 = list(range(4))     # [0,1,2,3]


# len, min, max, count, index
a = [9, 7, 9, 15, 12]
print(len(a))           # 5
print(min(a))           # 7
print(max(a))           # 15

print(a.count(9))       # 2     Indicates how often the 9 occurs
print(a.index(9))       # 0     Specifies the first index of the number 9
print(a.index(12))      # 4     Specifies the index of the number 12
```



all, any, in & not in

- all(seq)
  - Returns True if all elements of the sequence are True, otherwise False
- any(seq)
  - Returns True if one or more elements of the sequence True, otherwise False
- in & not in
  - Check whether an element occurs in the list or not

```
numbers = [9, 7, 9, 15, 12]
elem = 7
if elem in numbers:
   print("list contains element:", elem)
elem = 99
if elem not in numbers:
   print("list does not contain element:", elem)
```

Output

```
list contains element: 7
list does not contain element: 99
```


Read and assign values to elements

```
numbers = [1,2,3,4,5]
n = len(numbers)
print(type(numbers))
print(n)

print( numbers[0] )    # first element
print( numbers[-1] )   # last element
print( numbers[-2] )   # 2nd-last element
print( numbers[-n] )   # first element where n = len(numbers)

numbers[0] = 11        # change first element
print( numbers[0] )
```

Output

```
<class 'tuple'>
5

1
5
4
1

11
```

Add elements

Command	Description
li.append(elem)	Inserts the element elem at the end of the list li.
li.insert(pos, elem)	Inserts the element elem at position pos of the list li.
li.extend(sequence)	Inserts all elements of sequence at the end of the list li.

```
data.append("Hello")
data +=  ["word"]
print(data)

data.insert(1, "wonderful")
print(data)

data.extend(["Life", "is", "awesome"])
print(data)
```

Output

```
['Hello', 'word']
['Hello', 'wonderful', 'word']
['Hello', 'wonderful', 'word', 'Life', 'is', 'awesome']
```


Remove elements

Befehl	Beschreibung
li.pop()	Remove the last element and returned.
li.pop(pos)	Remove the element at position pos and returned.
li.remove(elem)	Remove the element elem.

```
items = [1, 2, 3, 4]
e = items.pop()
print(e)            # 4
print(items)        # [1, 2, 3]

e = items.pop(0)
print(e)            # 1
print(items)        # [2, 3]

items = [1, 2, 1, 7]
items.remove(1)
print(items)        # [2, 1, 7]
```


Slicing

```
numbers = (1,2,3,4,5)
print( numbers[:3] )          # first 3 elements
print( numbers[-2:] )         # last 2 elements

print( numbers[::2] )         # every 2nd element
print( numbers[1:10:2])       # every 2nd from 1 to 10
print( numbers[::] )          # all elements in sequence

first, second = numbers[:2]
print(first, second)
```

Slicing Syntax:
numbers[from : to : step]

Output

```
(1, 2, 3)
(4, 5)

(1, 3, 5)
(2, 4)
(1, 2, 3, 4, 5)

1 2
```


Slicing II

```
n = 20
r = list(range(n))
print("r       = ", r)
print("r[1:-2:3] = ",r[1:-2:3])

s = r[::3]
print("s       = ", s)
print("s[1:-2]   = ",s[1:-2])

print("r[::3][1:-2]",r[::3][1:-2])
```

Output

```
r          =  [0, 1, 2, 3, 4, 5, 6, 7, 8, ..., 17, 18, 19]
r[1:-2:3]  =  [1, 4, 7, 10, 13, 16]

s          =  [0, 3, 6, 9, 12, 15, 18]
s[1:-2]    =  [3, 6, 9, 12]

r[::3][1:-2] [3, 6, 9, 12]
```

sort

Command	Description
li.sort()	Sort the list li (reverse parameter controls the direction)
sorted(li)	Returns a sorted list (reverse parameter controls direction)

```
items = [1, 2, 0, 7]
items.sort()                    # [0, 1, 2, 7] sorts list 'in place', not return
items.sort(reverse = True)      # [7, 2, 1, 0]
sorted(items)                   # [0, 1, 2, 7]
sorted(items,reverse = True)    # [7, 2, 1, 0]

# Sort a String-List
words = "cheese, spam, egg, spam, bacon, spam"
sorted_words = sorted(words.split(", "))
sorted_words = ", ".join(sorted_words)
print(sorted_words)
```

Output

```
bacon, cheese, egg, spam, spam, spam
```


map

- map(function, iteratable)
  - Applies the function f to each element of the iterable
  - In Python 3 it returns a generator

```
gen = map(abs, [-1, 2, -3])
for el in gen:
   print(el)
```

Output

```
1
2
3
```

```
list( map(abs, [-1, 2, -3]) )           # [1, 2, 3]
list( map(min, [(1,2),(8,5)]) )         # [1, 5]
list( map(sorted, [(1,5,3), (8,5,2)]))  # [[1, 3, 5], [2, 5, 8]]
```

filter

- filter(function, iteratable)
  - keeps all elements el for which bool (function(el)) == True.
  - In Python 3 it returns a generator

```
def isGreaterZero(x):
  return x > 0;

print( list( filter(isGreaterZero, [-1,0,1,2]) ) )
```

Output

```
[1,2]
```


Lambda

- Defining a function for each filtering makes little sense
- The elegant solution is an anonymous lambda function
- Can be defined directly in a map or filter statement
- The anonymous lambda is actually assigned to a variable in the namespace of the function
- You can also assign a lambda function to a variable

Lambda II

```
# Filtern mit lambda:
data = [1, None, 2, 3]
print(data)

print (list(filter(lambda value: value is not None, data)))

notNone = lambda value : value is not None
print (list(filter(notNone, data)))
```

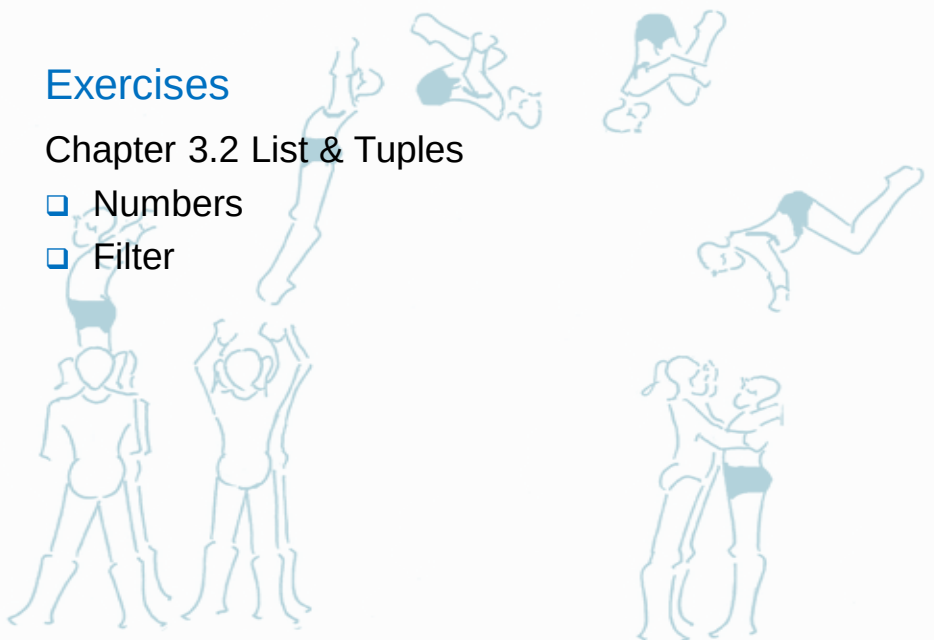
Output

```
[1, None, 2, 3]

[1, 2, 3]

[1, 2, 3]
```


Exercises

Chapter 3.2 List & Tuples

- Numbers
- Filter



Section III

Sets

Sets

- A set is an unordered collection of unique objects.

```
# Initialisierung
s = set()              # {}
s = set([1,2,3])       # {1, 2, 3}
s = set([1,2,3,1])     # {1, 2, 3}
s = set("sam")         # {'s','a','m'}

# Set aus Liste erstellen
l1 = [1, 2, 3, 2, 6, 2]
s1 = set(l1)           # {1, 2, 3, 6}
```

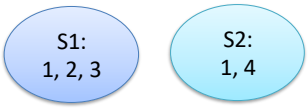
Doppelte Einträge fallen weg.

Add, Pop, Remove

```
# add
s = set([1,2,3])          # {1, 2, 3}
s.add(4)                  # {1, 2, 3, 4}
s.add(0)                  # {0, 1, 2, 3, 4}

# pop
e = s.pop()               # {1, 2, 3, 4}, e=0

# remove
s.remove(3)               # {1, 2, 4}
```


Union, Intersection, Difference

```
s1 = set([1,2,3])
s2 = set([1,4])

# union
s = s1.union(s2)          # s = {1, 2, 3, 4}

# intersection
s = s1.intersection(s2)   # s = {1}

# difference
s = s1.difference(s2)     # s = {2, 3}
s = s2.difference(s1)     # s = {4}
```



Example with loop

```
l = [1, 1, 2, 3, 2]
print(l)

s1 = set()
for e in l:
   s1.add(e)
print(s1)          # {1, 2, 3}

s2 = {e for e in l}
print(s2)          # {1, 2, 3}
```

Output

```
[1, 1, 2, 3, 2]
{1, 2, 3}
{1, 2, 3}
```


Section IV

Dictionaries

Dictionaries

- A dictionary contains key:value pairs and is not sorted
- Access is via key

```
# Initialisierung mit Literal
d1 = {}
d2 = {"key":"value", 1:"value 1", 2:"value 2"}

# Initialisierung mit Klasse
d1 = dict()
d2 = dict([ ("key","value of k"), (1, "value of 1"), (2, 4711) ])
d3 = dict([ ["key","value of k"], [1, "value of 1"], [2, 4711] ])

# Zugriff via key
print (d2["key"])       # value of k
print (d2[1])           # value of 1
print (d2[2])           # 4711
```


Dictionaries II

- Keys can be of different types (except list and dict, because mutable)
  - int, float, complex, bool, str, tuple, function

```
d = {
   (1,2) : "tuple value",
   print : "ok",
   "pi"   : 3.14159,
   4711  : "Kölnisch Wasser"
}
print( d[(1,2)] )
print( d[print] )
print( d["pi"] )
print( d[4711]
```

Output

```
tuple value
ok
3.14159
Kölnisch Wasser
```

Insert, Update, Merge, Delete

```
#  Assign new value
d = {1: "H"}
d[1] = "Hello"
print(d)                                # {1: 'Hello'}

# Add new element
d[2] = "World"
print(d)                                # {1: 'Hello', 2: 'World'}

# Update/Merge with other dict
dx = {2: "Tom", 3: "have a nice day"}
d.update(dx)                            # Update existing entries, insert new entries
print(d)                                # {1: 'Hello', 2: 'Tom', 3: 'have a nice day'}

# Remove element with pop(key)
d.pop(3)
print(d)                                # {1: 'Hello', 2: 'Tom'}
```


Keys, Values, Items

```
# Key, values, items:
d = {1:"v1", 2:"v2", 3:"v3"}
print(d.keys())                 # dict_keys([1, 2, 3])
print(d.values())               # dict_values(['v1', 'v2', 'v3'])
print(d.items())                # dict_items([(1, 'v1'), (2, 'v2'), (3, 'v3')])

# Iteration
d = {1:"v1", 2:"v2", 3:"v3"}

for k in d:
   print(k, "=", d[k])
```

Output

```
dict_keys([1, 2, 3])
dict_values(['v1', 'v2', 'v3'])
dict_items([(1, 'v1'), (2, 'v2'), (3, 'v3')])

1 = v1
2 = v2
3 = v3
```

Loops

```
d = {1:"v1", 2:"v2", 3:"v3"}
# Iterate over keys
for k in d.keys():
   print(k, "=", d[k])

# Iterate over values
for v in d.values():
   print(v)

# Iterate over items:
for k, v in d.items():
   print(k, "=", v)
```

Output

```
1 = v1
2 = v2
3 = v3
v1
v2
v3
1 = v1
2 = v2
3 = v3
```


87

Exercises

Chapter 3.3 Sets

- Mengen

Chapter 3.4 Dictionaries

- I18N
- Artikel

88

Chapter 4

Classes and objects

Classes & objects

- In Python, everything is an object: "hello".upper()
- Every class method needs the instances reference self as first parameter
  - The parameter name can be freely chosen
  - As convention self is used as name
- The type of the attributes is determined implicitly (as with all other variables)
- All constructors have the name __init__
- The destructor is called automatically by Python (garbage collection)

Classes & objects

```
class myClass:
   def __init__(self, arg1, arg2):
      self.var1 = arg1
      self.var2 = arg2

   def getVar1(self):
      return self.var1


# Create instance (object)
myInstance = myClass("p1", "p2")

# Invoke methode (self is passed implicitly)
result = myInstance.getVar1()
```


Constructor

- Types of constructors
  - Default constructor
  - Constructor with parameters
- Default constructor
  - Simple constructor without arguments
  - Only the instance reference self must be defined
- Constructor with parameters
  - Are called “parameterized constructors”
  - The first parameter is the instance reference self
  - The other parameters are defined by the developer

Default constructor

```
class GeekforGeeks:

    # default constructor
    def __init__(self):
        self.geek = "GeekforGeeks"

    # a method for printing data members
    def print_Geek(self):
        print(self.geek)

# creating object of the class
obj = GeekforGeeks()

# calling the instance method using the object obj
obj.print_Geek()
```

Instance reference 'self' must not be defined during the invocation

Output

```
GeekforGeeks
```


Constructor with parameters

```
class Addition:

    # parameterized constructor
    def __init__(self, f, s):
        self.first = f
        self.second = s
        self.answer = 0


    def display(self):
        print("First number = " + str(self.first))
        print("Second number = " + str(self.second))
        print("Addition of two numbers = " + str(self.answer))

    def calculate(self):
        self.answer = self.first + self.second

# creating object of the class, this will invoke parameterized constructor
obj = Addition(1000, 2000)

# perform Addition
obj.calculate()

# display result
obj.display()
```

Output

```
First number = 1000
Second number = 2000
Addition of two numbers = 3000
```

Destructor

- Destructors are called when an object instance is destroyed.
- They are the "counterpart" to the constructor.
- The destructor is called automatically by Python when the garbage collection destroys the instance.
- This can also be forced explicitly with the del command.

Destructor II

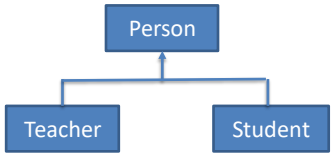
```
class Item:

   # constructor
   def __init__(self, number):
      self.number = number
      print("Create item:", self.number)

   # destructor
   def __del__(self):
      print("Delete item:", self.number)

# create instance
item = Item(47)

# delete instance
del item
```

Output

```
Create item: 47
Delete item: 47
```

Inheritance

```
class Person:
  def __init__(self, fname, lname):
    self.firstname = fname
    self.lastname = lname

  def printname(self):
    print(self.firstname, self.lastname)


class Teacher(Person):
  pass    # Use pass keyword when you do not want to
          # add any other properties or methods to the class.


class Student(Person):
  def __init__(self, fname, lname, studentId):
    super().__init__(fname, lname)
    self.studentId = studentId

  def printname(self):
    print( self.firstname, self.lastname,
           self.studentId)
```




```
x = Person("John", "Doe")
x.printname()

x = Teacher("Mike", "Olsen")
x.printname()

x = Student("James", "Bond", "007")
x.printname()
```

Output

```
John Doe
Mike Olsen
James Bond 007
```


Static attributes and methods

- It is also possible to define static attributes and methods
- These exists 1x and belong to the class
- They are called via class name, i.e. no instance is needed to access them
- The class attributes are defined on class level
- The methods are marked with @staticmethod

Mit @classmethod gibt es in Python noch eine weitere Abstufung.
Details siehe: https://rapd.wordpress.com/2008/07/02/python-staticmethod-vs-classmethod

Static attributes and methods II

```
class Counter:
    instance_count = 0

    def __init__(self):
        Counter.instance_count += 1

    @staticmethod
    def print(action):
        print(action, "instance. Count:", Counter.instance_count)

    def __del__(self):
        type(self).instance_count -= 1

if __name__ == "__main__":
    counters = []
    for i in range(4):
        counter = Counter()
        counters.append(counter)
        Counter.print("Created")
    while len(counters) > 0:
        counter = counters.pop()
        del counter
        Counter.print("Deleted")
```

Static attribute

Static methode

Output

```
Created instance. Count: 1
Created instance. Count: 2
Created instance. Count: 3
Created instance. Count: 4
Deleted instance. Count: 3
Deleted instance. Count: 2
Deleted instance. Count: 1
Deleted instance. Count: 0
```


Exercises

Chapter 4. Classes & objects

- Kreis
- Zylinder
- Fahrzeug

Optional Subject Unit Testing

- MathUtil
- UnitTest

Chapter 5

File Input/Output

File Input/Output

Command	Description
Open and close	
f = open("path/to/file", "r")	Open file to read
f = open("path/to/file", "w")	Open file to write
f = open("path/to/file", "a")	Open file to append
f.close()	Close file
Read	
s = f.read()	Read whole file content and return a string
s = f.readline()	Read next line of file and return string
l = f.readlines()	Read all lines of the file and return list
Write	
f.write("Hello World")	Write string to file
f.writelines[“Hello“, “World“])	Write list to file

Auto Close

- Open file and close it automatically at the end (of the with block):

```
with open("test-numbers.txt", "w") as f:
   f.writelines(["1.0\n", "1.5\n", "2.0\n"])

with open("hello.txt", "r") as f:
   text = f.read()

print(text)
```

Output

```
Hello World.
```


CSV files

```
import csv
titles = ["firstname", "lastname"]
rows   = [["Pipi", "Langstrumpf"], ["Peter", "Pan"], ["Marie", "Fischer"]]

with open("test.csv", "w") as f:
   writer = csv.writer(f)
   writer.writerow(titles)
   writer.writerows(rows)

data = []
with open("test.csv", "r") as f:
   reader = csv.reader(f)
   titles = reader.__next__()
   for row in reader:
      if (len(row) > 0):
         data.append(row)
print(data)
```

The csv module works with reader/writer objects

Output

```
[['Pipi', 'Langstrumpf'], ['Peter', 'Pan'], ['Marie', 'Fischer']]
```

CSV files with Pandas

```
import pandas

# read a csv file
with open("test.csv", "r") as f:
    data_frame = pandas.read_csv(f)

# write the data_frame to a csv file
data_frame.index.name = "index"
with open("test-pandas.csv", "w") as f:
    data_frame.to_csv(f)
```

test-pandas.csv

```
index,firstname,lastname
0,Pipi,Langstrumpf
1,Peter,Pan
2,Marie,Fischer
```

- In the chapter "Data Science Libraries" more information about the Pandas Framework will follow

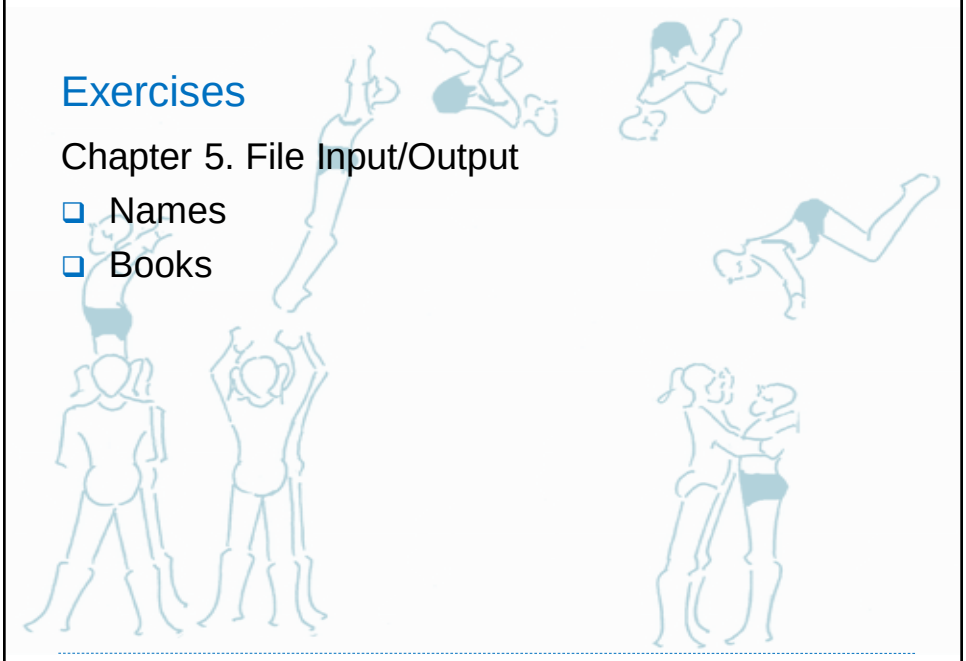
Read/Write objects

- Pickle can read and save Python objects:

```
import pickle

d = {1:2, "k":[1,2,3], "fun":print}
print(d)

with open("test-dict.pkl", "wb") as fout:      # open file for write-binary
    pickle.dump(d, fout)                       # dump pickle file

with open("test-dict.pkl", "rb") as fin:       # open file for read-binary
    d2 = pickle.load(fin)                      # load pickle file
print(d2)
```

Output

```
{1: 2, 'k': [1, 2, 3], 'fun': <built-in function print>}
{1: 2, 'k': [1, 2, 3], 'fun': <built-in function print>}
```



Exercises

Chapter 5. File Input/Output

- Names
- Books

Chapter 6

Modules & Packages

Section I

Modules

Modules

- With Python, definitions (functions, classes) can be swapped out into their own files (modules).
- The definitions of a module can be imported into other modules or the main program and used there.
- The file name corresponds to the module name with the suffix ".py".
- Within the module the module name is available via the internal variable "__name__".

Modul fibo.py

fibo.py

```
def print_fib(n):
    """ write Fibonacci series up to n """
    a, b = 0, 1
    while a < n:
        print(a, end=' ')
        a, b = b, a+b
    print()


def fib(n):
    """ return Fibonacci series up to n """
    a, b = 0, 1
    result = []
    while a < n:
        result.append(a)
        a, b = b, a+b
    return result
```


Usage of module fibo.py

```
import fibo

print ("Fibo sample:")
fibo.print_fib(100)

result = fibo.fib(100)
print(result)

print(("Show module details:"))
print(dir(fibo))
```

Output

```
Fibo sample:
0 1 1 2 3 5 8 13 21 34 55 89
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

Show module details:
['__builtins__', '__cached__', '__doc__', '__file__',
'__loader__', '__name__', '__package__', '__spec__', 'fib',
'print_fib']
```

Import

- Sample: import module
  - imports everything and keeps it in the module's namespace
  - module.func()
  - module.className.func()
- Sample: from module import *
  - imports everything under the current namespace
  - func()
  - className.func()
  - **not recommended**
- Sample: from module import className
  - selectively imports under the current namespace
  - className.func()
  - like standard modules: math, os, sys

Import with custom name

game.py

```
if visual_mode:
   # in visual mode, we draw using graphics
   import draw_visual as draw
else:
   # in textual mode, we print out text
   import draw_textual as draw

def main():
   result = play_game()
   # this can either be visual or textual depending on visual_mode
   draw.draw_game(result)

. . .
```

Section II

Packages

Packages

- Python modules are organized with the help of packages.
- Thereby a point notation is used.
- With the expression A.B, for example, module B is referenced within package A.
- Each module has its own namespace, this prevents name conflicts (of variables and classes).

Packages II

- Packages are directories, the modules of the packages are organized in corresponding subdirectories.
- Inside the package directory, a file __init__.py must be present for Python to identify the directory as a package.
- When a package is imported, Python searches the directories of the sys.path variable to locate the corresponding package directory.

Example package structure

```
sound/                          Top-level package
    __init__.py                 Initialize the sound package

    formats/                    Subpackage for file format conversions
      __init__.py
      wavread.py                > Module wavread
      wavwrite.py               > Module wavwrite
      ...
    effects/                    Subpackage for sound effects
      __init__.py
      echo.py                   > Module echo
      surround.py
      reverse.py
      ...
    filters/                    Subpackage for filters
      __init__.py
      equalizer.py
      vocoder.py
      karaoke.py
```

Import

- Import module and referencing with full name

```
import sound.effects.echo
sound.effects.echo.echofilter (input, output, delay=0.7, …)
```

- Import module without package prefix and reference with simple module name

```
from sound.effects import echo
echo.echofilter(input, output, delay=0.7, atten=4)
```

- Direct import of a function or variable from a module

```
from sound.effects.echo import echofilter
echofilter(input, output, delay=0.7, atten=4)
```


Import with *

- An import with *, import the modules specified in the package.
  - The specification is done in the file __init__.py with the help of the variable __all__.
- The variable defines a list of the modules that the package provides, such as:
  - __all__ = ["echo", "surround", "reverse"]
  - The following import statement would thus import the above modules at once:
    sound.effects import *

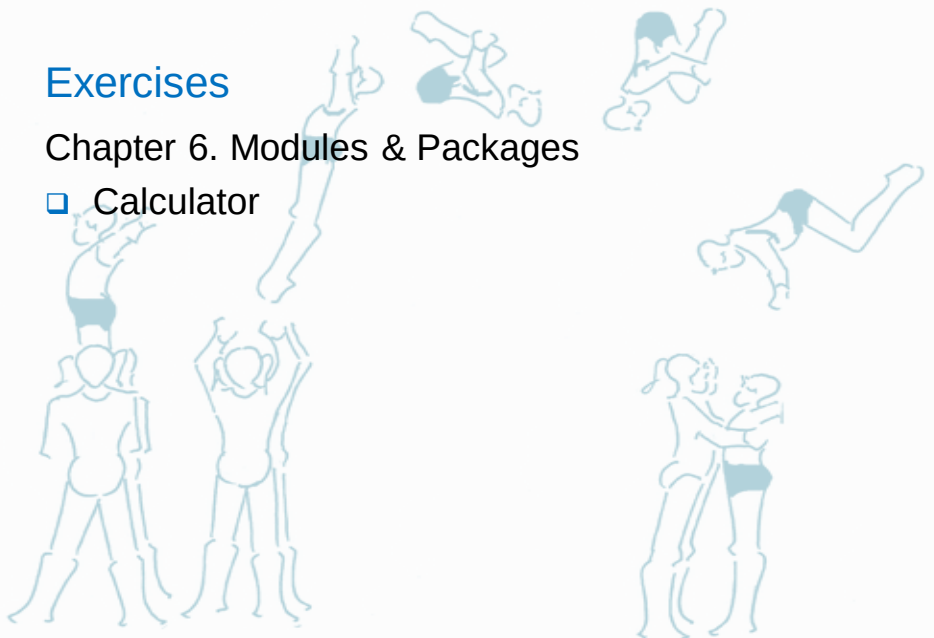
Import with * if __all__ is not defined

- It is the responsibility of the package developers to keep the list up-to-date with new package releases.
  - It is also possible not to specify anything (if you have no use for import with *)
- If __all__ is not defined, the sound.effects import * statement will not import the sound.effects modules.
  - It is only ensured that the package sound.effects is imported
  - The initialization code of the file __init__.py is executed and the names defined there are imported.
- Generally, imports with explicit specification of the modules are recommended.

Exercises

Chapter 6. Modules & Packages

- Calculator



Chapter 7

Standard Libraries

Section I

math

math

Methode	Beschreibung
math.acos()	Returns the arc cosine of a number
math.acosh()	Returns the inverse hyperbolic cosine of a number
math.asin()	Returns the arc sine of a number
math.asinh()	Returns the inverse hyperbolic sine of a number
math.atan()	Returns the arc tangent of a number in radians
math.atan2()	Returns the arc tangent of y/x in radians
math.atanh()	Returns the inverse hyperbolic tangent of a number
math.ceil()	Rounds a number up to the nearest integer
math.comb()	Returns the number of ways to choose k items from n items without repetition and order
math.copysign()	Returns a float consisting of the value of the first parameter and the sign of the second parameter

Quelle: https://www.w3schools.com/python/module_cmath.asp

math II

Methode	Beschreibung
math.cos()	Returns the cosine of a number
math.cosh()	Returns the hyperbolic cosine of a number
math.degrees()	Converts an angle from radians to degrees
math.dist()	Returns the Euclidean distance between two points (p and q), where p and q are the coordinates of that point
math.erf()	Returns the error function of a number
math.erfc()	Returns the complementary error function of a number
math.exp()	Returns E raised to the power of x
math.expm1()	Returns E^x - 1
math.fabs()	Returns the absolute value of a number
math.factorial()	Returns the factorial of a number
math.floor()	Rounds a number down to the nearest integer

math III

Methode	Beschreibung
math.log()	Returns the natural logarithm of a number, or the logarithm of number to base
math.log10()	Returns the base-10 logarithm of x
math.log1p()	Returns the natural logarithm of 1+x
math.log2()	Returns the base-2 logarithm of x
math.perm()	Returns the number of ways to choose k items from n items with order and without repetition
math.pow()	Returns the value of x to the power of y
math.prod()	Returns the product of all the elements in an iterable
math.radians()	Converts a degree value into radians
math.remainder()	Returns the closest value that can make numerator completely divisible by the denominator
math.sin()	Returns the sine of a number

127

math VI

Methode	Beschreibung
math.sin()	Returns the sine of a number
math.sinh()	Returns the hyperbolic sine of a number
math.sqrt()	Returns the square root of a number
math.tan()	Returns the tangent of a number
math.tanh()	Returns the hyperbolic tangent of a number
math.trunc()	Returns the truncated integer parts of a number

Konstanten	Beschreibung
math.e	Returns Euler's number (2.7182...)
math.inf	Returns a floating-point positive infinity
math.nan	Returns a floating-point NaN (Not a Number) value
math.pi	Returns PI (3.1415...)
math.tau	Returns tau (6.2831...)

128

cmath

- In addition to the Math functions shown, Python provides functions for complex numbers with the cmath module.
- The cmath methods support
  - int, float and complex numbers as well as
  - objects with __complex__() or __float__() methods.

Section II

os

os

- With the os module "Operation System" methods are supported.
- This includes for example methods to list or create files and directories.
- Or to execute shell commands:

```
# List files of current directory on linux/unix
os.system('ls -al')
```


os examples

```
# Changing the Current Working Directory
os.chdir(demo_dir)

print(os.getcwd())

# Create some files
open('fileA.txt', 'a').close()
open('fileB.txt', 'a').close()
open('fileC.txt', 'a').close()

# List Files and Sub-directories
print(os.listdir())
```

Output

```
..\example\07-std-libs\os-demo
['fileA.txt', 'fileB.txt', 'fileC.txt']
```

Section III

sys, subprocess

sys

- The sys module supports the execution of system commands as well as leaving the application with exit()
- Further more it is possible to list the loaded Python modules or get the version of the Python Interpreter
- The following table shows the most important comands in a quick overview

sys

Methode	Description
argv	Contains the script/program arguments. • Index 0 contains the script name • Then the arguments
executable	Output absolute path to the Python interpreter
exit	With exit, the program can be terminatd in a controlled manner, for example after an exception. • A code (usually an integer) can be specified. • The code 0 stands for a successful termination.
modules	Dictionary with all Python modules the interpreter has loaded since startup
path	List with directory paths in which modules are searched for. • The initialization is based on the environment variable PYTHONPATH. • With append further directories (paths) can be appended

Quelle: https://python101.pythonlibrary.org/chapter20_sys.html

sys

Methode	Beschreibung
platform	Information of the platform identifier ('win32', etc.). Can be used for platform specific code (imports for example)
stdin / stdout / stderr	Standard input, output and error streams of the interpreter.
version	Output the version of the Python interpreter

Example with sys.platform

```
os = sys.platform
if os == "win32":
   # use Window-related code here
   import _winreg
elif os.startswith('linux'):
   # do something Linux specific
   import subprocess
   subprocess.Popen(["ls, -l"]
```

subprocess

- With subprocess processes or other programs can be started from Python
- The module was introduced with Python 2.4 as a replacement for os.popen, os.spawn and os.system

Example

```
import subprocess

code = subprocess.call("notepad.exe")

if code == 0:
   print("Success!")
else:
   print("Error!")
```


Section IV

re (Regular Expression)

Regular Expression

- A regular expression is a sequence of characters that define a pattern.
- This can be used to search strings for patterns or check if they match a certain format, such as:
  - Credit card format
  - AHV number
  - ISBN number
  - e-mail address
- In Python the module re is used for this

Methods

- The re module contains the following methods:

Methode	Beschreibung
findall	Returns a list containing all matches
search	Returns a Match object if there is a match anywhere in the string
split	Returns a list where the string has been split at each match
sub	Replaces one or many matches with a string

- The Match object has properties and methods used to retrieve information about the search, and the result:
  - span() returns a tuple containing the start-, and end positions of the match.
  - string returns the string passed into the function
  - group() returns the part of the string where there was a match

Quelle: https://www.w3schools.com/python/python_regex.asp

findall()

```
# Print a list of all matches:
txt = "The rain in Spain"
x = re.findall("ai", txt)

print(x)

# Return an empty list if no match was found:
txt = "The rain in Spain"
x = re.findall("Portugal", txt)

print(x)
```

Output

```
['ai', 'ai']
[]
```


search()

```
# Search for the first white-space character in the string:
txt = "The rain in Spain"
x = re.search("\s", txt)

print("The first white-space character is located in position:", x.start())

# Make a search that returns no match:
txt = "The rain in Spain"
x = re.search("Portugal", txt)

print(x)
```

Output

```
The first white-space character is located in position: 3
None
```

split()

```
# Split at each white-space character:
txt = "The rain in Spain"
x = re.split("\s", txt)

print(x)

# Split the string only at the first occurrence:
txt = "The rain in Spain"
x = re.split("\s", txt, 1)

print(x)
```

Output

```
['The', 'rain', 'in', 'Spain']
['The', 'rain in Spain']
```


sub()

```
# Replace every white-space character with the number 9:
txt = "The rain in Spain"
x = re.sub("\s", "9", txt)

print(x)

# Replace the first 2 occurrences:
txt = "The rain in Spain"
x = re.sub("\s", "9", txt, 2)

print(x)
```

Output

```
The9rain9in9Spain
The9rain9in Spain
```

Meta Character

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"world$"
*	Zero or more occurrences	"aix*"
+	One or more occurrences	"aix+"
{}	Exactly the specified number of occurrences	"al{2}"
|	Either or	"falls|stays"
()	Capture and group	

Special Sequences

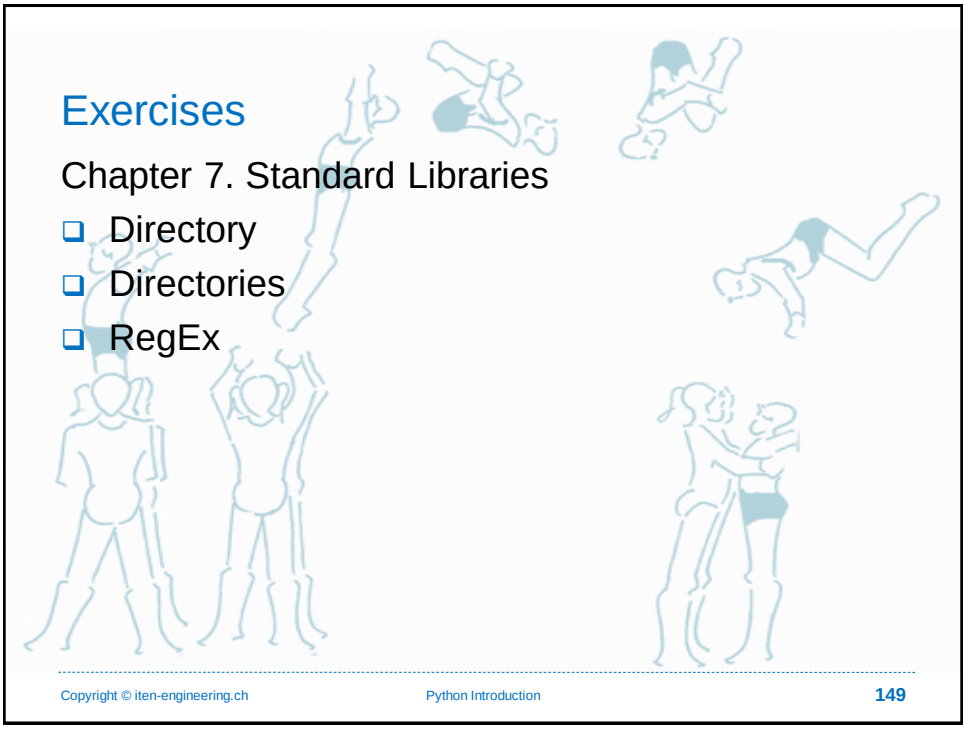
Character	Description	Example
\A	Returns a match if the specified characters are at the **beginning** of the string	"\AThe"
\b	Returns a match where the specified characters are at the **beginning or at the end of a word** (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\bain" r"ain\b"
\B	Returns a match where the specified characters are **present, but NOT at the beginning (or at the end)** of a **word** (the "r" in the beginning is making sure that the string is being treated as a "raw string")	r"\Bain" r"ain\B"
\d	Returns a match where the string **contains digits** (numbers from 0-9)	"\d"
\D	Returns a match where the string **DOES NOT contain digits**	"\D"

Special Sequences II

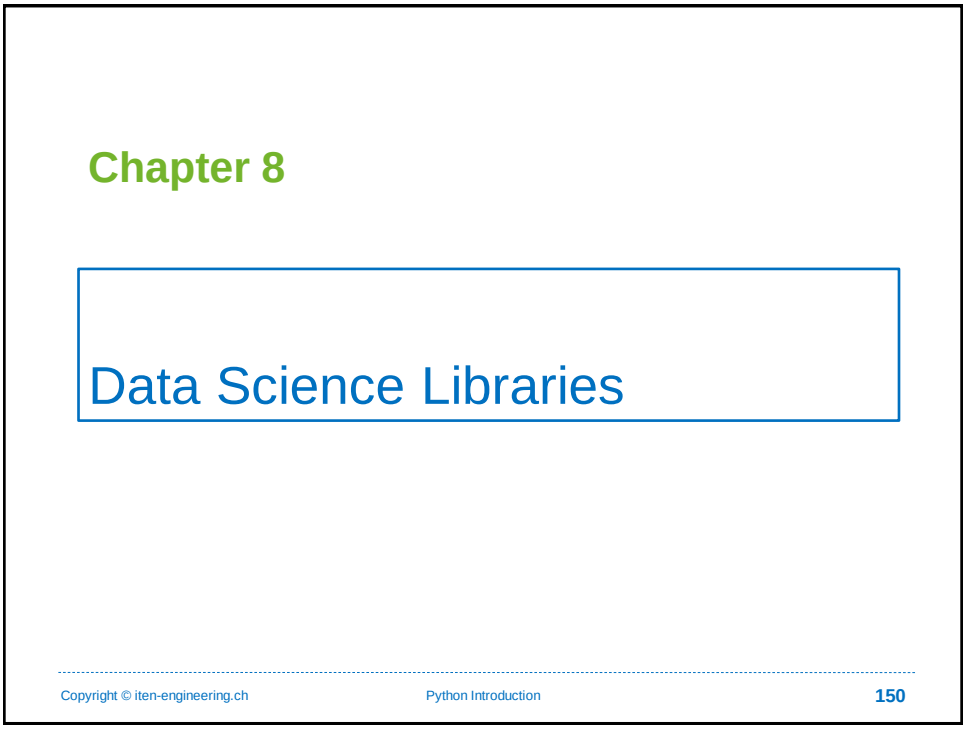
Character	Description	Example
\s	Returns a match where the string **contains** a **white space** character	"\s"
\S	Returns a match where the string **DOES NOT contain** a **white space** character	"\S"
\w	Returns a match where the string contains **any word** characters (characters from a to Z, digits from 0-9, **and the underscore** _ character)	"\w"
\W	Returns a match where the string **DOES NOT contain any word characters**	"\W"
\Z	Returns a match if the specified characters are at **the end of the string**	"Spain\Z"

Sets

Set	Description
[arn]	Returns a match where **one of the specified characters** (a, r, or n) are **present**
[a-n]	Returns a match for any lower case **character**, alphabetically between **a and n**
[^arn]	Returns a match for any character **EXCEPT a, r, and n**
[0123]	Returns a match where any of the **specified digits** (0, 1, 2, or 3) are **present**
[0-9]	Returns a match for any **digit between 0 and 9**
[0-5][0-9]	Returns a match for any **two-digit numbers from 00 and 59**
[a-zA-Z]	Returns a match for any character alphabetically between **a and z, lower case OR upper case**
[+]	In sets, +, *, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string


Exercises
Chapter 7. Standard Libraries
Directory
Directories
RegEx
Copyright © iten-engineering.ch
Python Introduction
149

149


Chapter 8
Data Science Libraries
Copyright © iten-engineering.ch
Python Introduction
150

150

Section I

Overview

Data Science Libraries

- Python is (besides R) for many the first choice for Data Science tasks.
- There are a variety of libraries for the different statistics and data science areas

Topic	Libraries
Data Mining	Scrapy, Bautiful Soap
Data Processing and Modeling	NumPy, SciPy, Pandas, Keras, SciKit Learn, PyTorch, TensorFlow, XGBoost
Data Visualization	Mathplotlib, Seaborn, Bokeh, Plotly, pydot

NumPy

→ Details Section II

- NumPy stands for Numerical Python and is a perfect tool for scientific work.
- The library offers many practical features for operations with arrays, storing values of the same data type
- Mathematical operations with NumPy arrays (as well as their vectorization) are efficient and powerful
- Many other libraries are based on NumPy

SciPy

- SciPy is a collection of mathematical functions and algorithms and is based on NumPy.
- It provides efficient routines for optimization, integration and other tasks
- SciPy is organized in sub packages, which are usually imported and used individually

```
import numpy as np
from scipy import optimize

x = np.arange(0,10)
y = 2 * x + 3 + np.random.random(10)
res = optimize.curve_fit(lambda x, a, b: a*x + b, x, y)
```

SciPy Packages

- Special functions (scipy.special)
- Integration (scipy.integrate)
- Optimization (scipy.optimize)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fft)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)
- Statistics (scipy.stats)
- Image processing (scipy.ndimage)
- File IO (scipy.io)
- Sparse eigenvalue with ARPACK (scipy.sparse)
- Compressed Sparse Graph (scipy.sparse.csgraph)
- Spatial data and algorithms (scipy.spatial)

Reference: https://docs.scipy.org/doc/scipy/reference

Pandas

→ Details Section III

- Pandas is the Python module for data analysis and manipulation and is also based on NumPy
  - merge data sets
  - group data (i.e. split into groups)
  - perform operations on groups
  - Label-based indexing and slicing
- Provides two classes:
  - 1-dimensional data (Series)
  - 2-dimensional data with labels (DataFrame)

SciKit Learn

- The name SciKit originated from the words SciPy and Toolkit
- SciKit Learn is based on SciPy and delivers machine learning functions for:
  - Supervised-learning (Classification, Regression)
  - Unsupervised-learning (Clustering, Density-estimation)
  - Data preparation (preprocessing, feature extraction, feature selection)
  - Estimator scoring

Keras

- Keras is an open source library in the field of deep learning.
- Neural networks can be modeled and trained
- Keras offers a uniform interface to other frameworks such as TensorFlow, Microsoft Cognitive Toolkit or Theano.
- In future releases, Keras will be primarily aligned with TensorFlow.

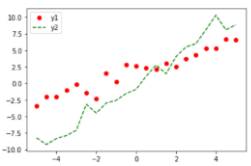
TensorFlow

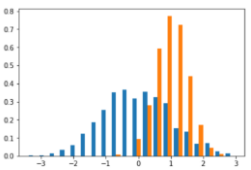
- TensorFlow is a Python framework for Machine Learning and Deep Learning.
- It is the best tool for the topics of object identification and speech recognition
- It helps in working with artificial neural networks that need to process multiple data sets.
- TensorFlow is constantly being enhanced, for example like
  - Fixes to potential security vulnerabilities
  - Improvements in the integration of GPU support

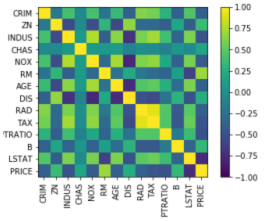
Mathplotlib

→ Details see exercises

- The matplotlib library is the Python module for graphing data.
- It is similar to the graphical functions of Mathlab
- The module matplotlib.pyplot contains the functions to create graphs







Section II

NumPy

NumPy

- The NumPy library provides basic objects (array and matrix) for technical computing in Python
- It has functions for mathematics, statistics and linear algebra
- The import is usually done with the abbreviation np
- Numpy is often used as basis for other libraries (like Pandas)

```
import numpy as np

b = np.array([6, 7, 8])
print(b)                    # array([6, 7, 8])
print(type(b))              # <class 'numpy.ndarray'>
```

Array information

Methode	Description
ndim	The number of axes (dimensions) of the array
shape	Tuple of integers indicating the size of the array in each dimension. • for a matrix with n rows and m columns, shape will be (n,m). • the length of the shape tuple is therefore the number of axes
size	Total number of elements of the array.
dtype	Object describing the type of the elements in the array. • The type can be specified (Standard Python or Numpy types).
itemsize	The size in bytes of each element of the array. • For example, an array of elements of type float64 has itemsize 8 (=64/8), while one of type complex32 has itemsize 4 (=32/8). • It is equivalent to ndarray.dtype.itemsize.
data	The buffer containing the actual elements of the array. • Normally, we won't need to use this attribute because we will access the elements in an array using indexing facilities.
reshape	reshape array elements to the given shape

Array information II

```
a = np.arange(15).reshape(3, 5)
print(a)            # [[ 0,  1,  2,  3,  4],      Zweidimensionales Array ndim = 2
                    #  [ 5,  6,  7,  8,  9],      mit 3 à 5 Elemente shape = (3, 5)
                    #  [10, 11, 12, 13, 14]]
print(a.shape)      # (3, 5)
print(a.ndim)       # 2
print(a.dtype.name) # 'int64'
print(a.itemsize)   # 8
print(a.size)       # 15
print(type(a))      # <class 'numpy.ndarray'>
```

Output

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]]
(3, 5)
2
int32
4
15
<class 'numpy.ndarray'>
```


Array creation

```
a = np.array(1,2,3,4)                # WRONG,  Traceback (most recent call last):
                                     # TypeError: array() takes from 1 to 2 positional arguments
                                     # but 4 were given

a = np.array([1,2,3,4])              # RIGHT
print(a)

b = np.array([(1.5,2,3), (4,5,6)])   # [[1.5, 2. , 3. ],
print(b)                             #  [4. , 5. , 6. ]]

# The type of the array can also be explicitly specified at creation time:
c = np.array( [ [1,2], [3,4] ], dtype=complex )

print(c)                             # [[1.+0.j, 2.+0.j],
                                     #  [3.+0.j, 4.+0.j]]
```


Array creation II

- Often, the elements of an array are originally unknown, but its size is known.
- Hence, NumPy offers several functions to create arrays with initial placeholder content.
- These minimize the necessity of growing arrays, an expensive operation.
  - zeros: the function zeros creates an array full of zeros,
  - ones: the function ones creates an array full of ones,
  - empty: the function empty creates an array whose initial content is random and depends on the state of the memory. By default, the dtype is float64.
  - arrange: the function arrange creates a range: from, to, step (the to is exclusive)

Array creation III

```
x = np.zeros((3, 4))
print(x)

y = np.ones( (2,3,4), dtype=np.int16 )
print(y)

z = np.empty( (2,3) )
print(z)

a = np.arange(0.0, 1.01, 0.1)     # [0.0 0.1 0.2 … 0.9 1.0]
print(a)
```

Output

```
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
[[1 1 1 1]
 [1 1 1 1]
 [1 1 1 1]]
[[1.5 2.  3. ]
 [4.  5.  6. ]]
[0.  0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1. ]
```


Random numbers

- The random function can be used to initialize arrays with random numbers:

```
x = np.random.random(3)      # Numbers from 0..1
print(x)
                             # Numbers from normal distribution
y = np.random.normal(loc=0.0, scale=1.0, size=10)
print(y)
                             # Numbers from the given list
z = np.random.choice([2,4,6,8], 10)
print(z)
```

Output

```
[0.80217922 0.41181568 0.98317354]

[ 0.17643768  0.13406789  0.62499216 -0.99067793  0.72721655 -1.18774602
  0.03740755  1.23494155 -0.7377875   0.21212164]

[2 2 6 8 6 6 4 6 4 8]
```


Array slicing

```
x = np.array(
   [[1, 2, 3],
    [4, 5, 6],
    [7, 8, 9]])

print(x[0, :])        # first line
print(x[:, 0])        # first column
print(x[0, 0])        # first element of first column
print(x[0:2,1])       # first 2 elements of 2nd column
```

Output

```
[1 2 3]
[1 4 7]
1
[2 5]
```


Onedimensional array slicing

```
a = np.arange(10)**3
print(a)              # a = [0 1 8 27 64 125 216 343 512 729]
print(a[2])           # 8
print(a[2:5])         # [8 27 64]

a[0:6:2] = 1000       # from postion 0 to 6 exclusive, set every 2nd element to 1000
                      # (equivalent to: a[:6:2] = 1000)
                      # [1000,   1, 1000,  27, 1000,  125,  216,  343,  512,  729]

a[ : :-1]             # reversed a
                      # [ 729,  512,  343,  216,  125, 1000,   27, 1000,    1, 1000]
```

Output

```
[   0    1    8   27   64  125  216  343  512  729]
8
[ 8 27 64]
```

Multidimensional array slicing

```
def myfunc(x,y):
  return 10*x + y

b = np.fromfunction(myfunc,(5,4),dtype=int)    # Initialisierung: x = 0...4 mit jeweils y = 0..3

print(a)                  # [ [ 0,  1,  2,  3],
                          #   [10, 11, 12, 13],
                          #   [20, 21, 22, 23],
                          #   [30, 31, 32, 33],
                          #   [40, 41, 42, 43] ]

print(b[2,3])             # 23

print(b[0:5, 1])          # each row in the second column of b (equivalent to: b[ : ,1])
                          # [ 1, 11, 21, 31, 41]

print(b[1:3, : ])         # each column in the second and third row of b
                          # [ [10, 11, 12, 13],
                          #   [20, 21, 22, 23] ]
```


Operations per element

```
# Comparsion (element-wise)
a = np.array([1,2,3,4,5])

b = a > 3                 # "element-wise" compare creates an array with boolean
print(b)                  # b = [False False False True True]

c = a[b]                  # Boolean array as mask
print(c)                  # c = [4 5]

d = a[a <= 3]             # Boolean array as mask
print(d)                  # d = [1 2 3]
```

Output

```
[False False False  True  True]
[4 5]
[1 2 3]
[25 35 45 55]
[ 4.  6.  8. 10.]
```

Operations per element II

```
# Operations like [+, -, *, /]
a = np.array([20,30,40,50] )

b = a + 5
print(b)

c = a / 5
print(c)

# Arithmetic operators on arrays apply elementwise.
a = np.array([20,30,40,50] )
b = np.array([10,15,20,25] )
c = a-b
print(c)
```

Output

```
[25 35 45 55]
[ 4.  6.  8. 10.]
[10 15 20 25]
```


nan, any and logical functions

```
x  = np.nan                 # undefined element
y = np.isnan(x)             # Test for undefined element
print(y)                    # y = True

a = np.array([1,2,np.nan,4,5])
b= np.isnan(a)              # Test "element-wise" for undefined element
print(b)                    # b = [False False True False False]

c = np.any(b)               # Check if at least one entry is True
print(c)                    # c = True

d = np.logical_not(b)       # logical functions (logical_and/not/or/xor)
print(d)
```

Output

```
True
[False False  True False False]
True
[ True  True False  True  True]
```

Statistics

```
a = np.array([7, 9, 4, 12, 6, 4])

print(np.min(a))
print(np.max(a))
print(np.mean(a))          # Mean value
print(np.std(a))           # Standard deviation
print(np.sort(a))
```

Output

```
4
12
7.0
2.8284271247461903
[ 4  4  6  7  9 12]
```


Section III

Pandas

Pandas

- The Pandas DataFrame is similar to the data.frame class in R
- 2-dimensional data with labels for column/row
- Implemented as wrapper around numpy.ndarray
  - Operations like with numpy arrays
  - You can usse masks (for indexing)
- A DataFrame is instantiated via the corresponding class

DataFrame

```
import numpy as np
import pandas as pd

col_names = ["A", "B", "C"]
row_names = ["First", "Second", "Third"]
data = [[1, 2, 3],
     [4, 5, 6],
     [7, 8, 9]]
df = pd.DataFrame(data, row_names, col_names)
print(df)

print(df.shape)      # Tuple with number of rows and columns
print(df.shape[0])   # Number of rows
print(df.shape[1])   # Number of columns
```

```
        A  B  C
First   1  2  3
Second  4  5  6
Third   7  8  9

(3, 3)
3
3
```

Output

Slicing

```
# DataFrame
print(df)

# Spalten selektieren
c1 = df["A"]
print(c1)

c2 = df.B
print(c2)
```

Output

```
         A  B  C
First    1  2  3
Second   4  5  6
Third    7  8  9


First     1
Second    4
Third     7
Name: A, dtype: int64


First     2
Second    5
Third     8
Name: B, dtype: int64
```


Slicing II

```
# DataFrame
print(df)

# Zeilen selektieren
# Label slicing ist inklusive End Label
r1_r2 = df["First":"Second"]
print(r1_r2)

# Slicing via Index ist exklusive End Index
r2 = df[1:2]
print(r2)

# Das loc Attribut erlaubt Label
# basiertes Slicing von Kolone und Zeile
x = df.loc["First":"Second", "B":"C"]
print(x)
```

Output

```
         A  B  C
First    1  2  3
Second   4  5  6
Third    7  8  9

         A  B  C
First    1  2  3
Second   4  5  6

         A  B  C
Second   4  5  6



         B  C
First    2  3
Second   5  6
```

apply

- The apply method allows a function to be applied to any element or series of elements of a DataFrame:

```
sq = df.apply(np.sqrt)
print(sq)


min_per_col = df.apply(min)
print(min_per_col)


def mycalc(x):
  return 2*x
mc = df.apply(mycalc)
print(mc)
```

```
              A         B         C
First  1.000000  1.414214  1.732051
Second 2.000000  2.236068  2.449490
Third  2.645751  2.828427  3.000000

A    1
B    2
C    3
dtype: int64

df
        A  B  C
First   1  2  3
Second  4  5  6
Third   7  8  9

        A   B   C
First   2   4   6
Second  8  10  12
Third  14  16  18
```

Output

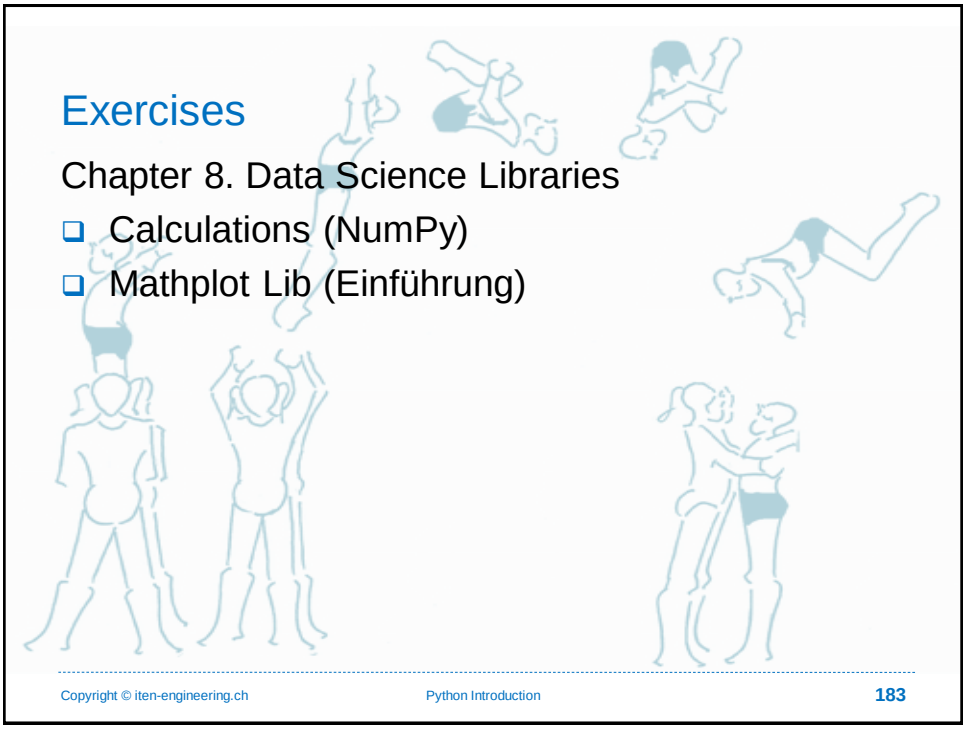
groupby

- With the groupby method data is grouped by a column
- You can then apply functions per group on the returned object

```
col_names = ["A", "B"]
data = [[1, 2],
        [1, 4],
        [2, 4],
        [1, 2]]
df = pd.DataFrame(data, columns=col_names)

grouped = df.groupby("A")  # 1: 2 4 2 : mean = 8/3 = 2.67
                           # 2: 4     : mean = 4/1 = 4
print(grouped.mean())
```

```
          B
A
1  2.666667
2  4.000000
```

Output



Chapter 9

Application examples

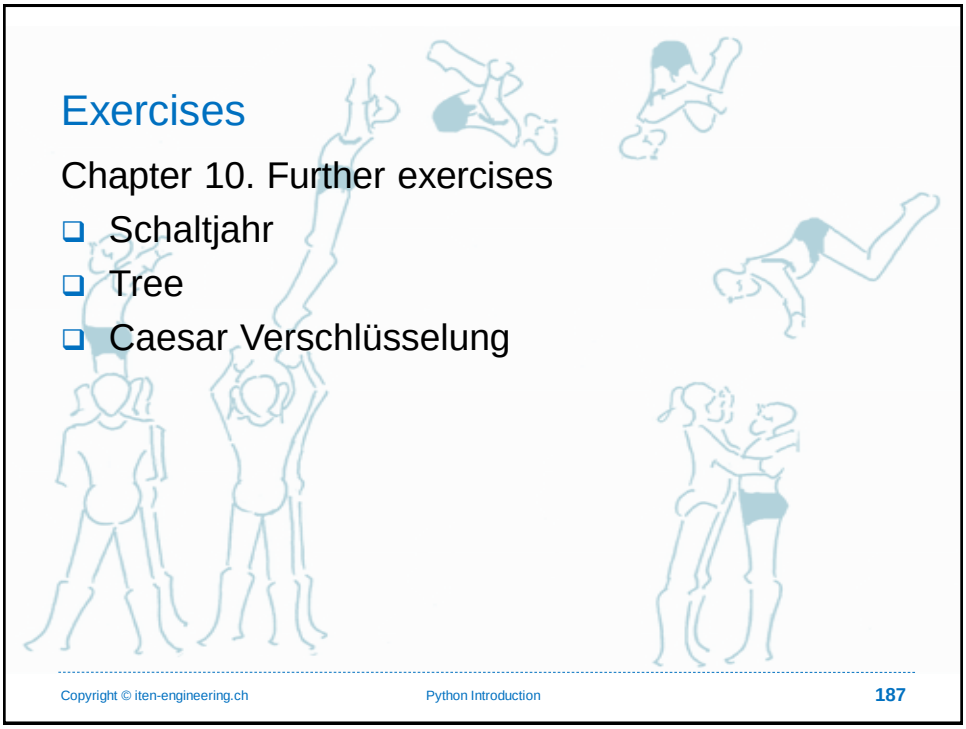
Exercises

Chapter 9. Application examples

- SciKitLearn
- Bookservice
- Streamlit
  - HelloStreamlit
  - Sales

Chapter 10

Further exercises



Exercises

Chapter 10. Further exercises

- Schaltjahr
- Tree
- Caesar Verschlüsselung

Chapter 11

Literature and web links

Literatur und Weblinks

- Python Home
  https://www.python.org
- Python Docs
  https://docs.python.org/3
- NumPy
  https://numpy.org
- SciPy
  https://www.scipy.org
- Pandas
  https://pandas.pydata.org
- SciKit Learn
  https://scikit-learn.org/stable

Chapter 12

Appendix

Reserved Words

and	assert	break	class
continue	def	del	elif
else	except	exec	finally
for	from	global	if
import	in	is	lambda
not	or	pass	print
raise	return	try	While
with			

Escape Sequence

Escape-sequence	Purpose
\n	New line
\\	Backslash character
\'	Apostrophe '
\"	Quotation mark "
\a	Sound signal
\b	Slaughter (backspace key symbol)
\f	The conversion of format
\r	Carriage return
\t	Horizontal tab
\v	Vertical tab
\x*hh*	Character with hex code hh
ooo	Character with octal value ooo
\0	Character Null (not a string terminator)
\N{*id*}	Identifier ID of Unicode database
\u*hhhh*	16-bit Unicode character in hexadecimal format
\U*hhhhhhhh*	32-bit Unicode character in hexadecimal format

Unpack arguments

- Arguments "unpack" with *args and **kwargs
  - *list unpacks elements of a list
  - **dict unpacks elements of a dictionary

```
def f4(a, b, c=None, d=None):
  print("f4: a={}, b={}, c={}, d={}".format(a,b,c,d))

f4(*[1,2])                    # Unpack elements from a list
f4(*[1,2], d=4)

f4(1, 2, **{"c":3, "d":4})    # Unpack elements from a dict
f4(1, 2, **{"d":4, "c":3})
```

Output

```
f4: a=1, b=2, c=None, d=None
f4: a=1, b=2, c=None, d=4
f4: a=1, b=2, c=3, d=4
f4: a=1, b=2, c=3, d=4
```


Function with variable number of arguments

- Function definition with * and **:
  - Conventionally one uses *args and **kwargs
  - args contains all additional "positional" arguments
  - kwargs contains additional "keyword" arguments

```
def f5(a, *args, k=9, **kwargs):
  print("f5: a={}, args={}, k={}, kwargs={}".format(a,args,k,kwargs))

f5(1)
f5(1,2,4,6,k=7,x=9,y=11)
```

Output

```
f5: a=1, args=(), k=9, kwargs={}
f5: a=1, args=(2, 4, 6), k=7, kwargs={'x': 9, 'y': 11}
```

Multiple return values

- Python functions can return multiple variables.
  - These variables can be stored in variables directly.
  - A function is not required to return a variable, it can return zero, one, two or more variables.
- This is a unique property of Python, other programming languages such as C++ or Java do not support this.

```
def getPerson():
    name = "Leona"
    age = 35
    country = "UK"
    return name,age,country

name,age,country = getPerson()
print(name)
print(age)
print(country)
```

Quelle: https://pythonbasics.org/multiple-return

Built-in Functions

		Built-in Functions		
abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

Quelle: https://docs.python.org/3.3/library/functions.html

Built-in Functions - zip

- Make an iterator that aggregates elements from each of the iterables

```
>>> x = [1, 2, 3]
>>> y = [4, 5, 6]
>>> zipped = zip(x, y)
>>> list(zipped)
[(1, 4), (2, 5), (3, 6)]
>>> x2, y2 = zip(*zip(x, y))
>>> x == list(x2) and y == list(y2)
True
```

Quelle: https://docs.python.org/3.3/library/functions.html#zip

Frozen Sets

- The frozenset() function returns an immutable frozenset initialized with elements from the given iterable.
- If no parameters are passed, it returns an empty frozenset.

```
# tuple of vowels
vowels = ('a', 'e', 'i', 'o', 'u')

fSet = frozenset(vowels)
print('The frozen set is:', fSet)
print('The empty frozen set is:', frozenset())

# frozensets are immutable
fSet.add('v')
```

Output

```
The frozen set is: frozenset({'a', 'o', 'u', 'i', 'e'})
The empty frozen set is: frozenset()
Traceback (most recent call last):
  File "<string>, line 8, in <module>
    fSet.add('v')
AttributeError: 'frozenset' object has no attribute 'add'
```

Quelle: https://www.programiz.com/python-programming/methods/built-in/frozenset

Shallow and deep copy

- The difference between shallow and deep copying is only relevant for compound objects (objects that contain other objects, like lists or class instances):
  - A shallow copy constructs a new compound object and then (to the extent possible) inserts references into it to the objects found in the original.
  - A deep copy constructs a new compound object and then, recursively, inserts copies into it of the objects found in the original.

Quelle: https://docs.python.org/3/library/copy.html

Shallow and deep copy II

- Two problems often exist with deep copy operations that don't exist with shallow copy operations:
  - Recursive objects (compound objects that, directly or indirectly, contain a reference to themselves) may cause a recursive loop.
  - Because deep copy copies everything it may copy too much, such as data which is intended to be shared between copies.
- The deepcopy() function avoids these problems by:
  - keeping a memo dictionary of objects already copied during the current copying pass; and
  - letting user-defined classes override the copying operation or the set of components copied.

Quelle: https://docs.python.org/3/library/copy.html

Shallow and deep copy III

```
import copy

print("# Copy with: =")
old = [[1, 2, 3], [4, 5, 6], [7, 8, 'a']]
new = old
new[2][2] = 9
print('Old list:', old, "with ID", id(old) )
print('New list:', old, "with ID", id(new) )

print("# Copy with: copy")
old = [[1, 2, 3], [4, 5, 6], [7, 8, 'a']]
new = copy.copy(old)
new[2][2] = 9
print('Old list:', old, "with ID", id(old) )
print('New list:', old, "with ID", id(new) )

print("# Copy with: deepcopy")
old = [[1, 2, 3], [4, 5, 6], [7, 8, 'a']]
new = copy.deepcopy(old)
new[2][2] = 9
print('Old list:', old, "with ID", id(old) )
print('New list:', old, "with ID", id(new)
```

Output

```
# Copy with: =
Old list: [[1, 2, 3], [4, 5, 6], [7, 8, 9]] with ID 2204789025864
New list: [[1, 2, 3], [4, 5, 6], [7, 8, 9]] with ID 2204789025864

# Copy with: copy
Old list: [[1, 2, 3], [4, 5, 6], [7, 8, 9]] with ID 2204821800712
New list: [[1, 2, 3], [4, 5, 6], [7, 8, 9]] with ID 2204838633032

# Copy with: deepcopy
Old list: [[1, 2, 3], [4, 5, 6], [7, 8, 'a']] with ID 2204837090376
New list: [[1, 2, 3], [4, 5, 6], [7, 8, 'a']] with ID 2204821800712
```


Operator Overloading

Operator	Method
+	__add__(self, other)
-	__sub__(self, other)
*	__mul__(self, other)
/	__truediv__(self, other)
%	__mod__(self, other)
<	__lt__(self, other)
<=< code>	__le__(self, other)
==	__eq__(self, other)
!=	__ne__(self, other)
>	__gt__(self, other)
>=	__ge__(self, other)

Quelle: https://stackabuse.com/overloading-functions-and-operators-in-python

Python debugger

- The Python debugger module is called pdb and provides an interactive source code debugger.
- You can set breakpoints, execute the code step by step, inspect the stack frames and more.
- In an integrated development environment you can debug in the usual way and don't have to care about the pdb module.